[MAC0211] Laboratório de Programação I
Aula 5
Linguagem de Montagem
(Exemplos de Programas)

Kelly, adaptado por Gubi

DCC–IME–USP

14 de agosto de 2017

"Hello, world!" para NASM (versão 32 bits) – hello.asm

```
global _start       ; exporta para o ligador (ld) o ponto de entrada

section .text
_start:

    ; sys_write(stdout, mensagem, tamanho)

    mov eax, 4            ; chamada de sistema sys_write
    mov ebx, 1            ; stdout
    mov ecx, mensagem     ; endereço da mensagem
    mov edx, tamanho      ; tamanho da string de mensagem
    int 80h               ; chamada ao núcleo (kernel)

    ; sys_exit(return_code)

    mov eax, 1          ; chamada de sistema sys_exit
    mov ebx, 0          ; retorna 0 (sucesso)
    int 80h             ; chamada ao núcleo (kernel)

section .data
mensagem: db 'Hello, world!',0x0A    ; mensagem e quebra de linha
tamanho:  equ $ - mensagem           ; tamanho da mensagem
```

"Hello, world!" para GAS (versão 32 bits) – hello.S

```
.text

   .global  _start     ; exporta para o ligador (ld) o ponto de entrada

_start:

    # sys_write(stdout, mensagem, tamanho)
    movl    $4, %eax              # chamada de sistema sys_write
    movl    $1, %ebx              # stdout
    movl    $mensagem, %ecx       # endereço da mensagem
    movl    $tamanho, %edx        # tamanho da string de mensagem
    int     $0x80                 # chamada ao núcleo (kernel)

    # sys_exit(codigo_retorno)
    movl    $1, %eax       # chamada de sistema sys_exit
    movl    $0, %ebx     # retorna 0 (sucesso)
    int     $0x80        # chamada ao núcleo (kernel)

.data
mensagem:
    .ascii  "Hello, world!\n"     # mensagem e quebra de linha
    tamanho =    . - mensagem     # tamanho da mensagem
```

"Hello, world!" para NASM (versão 64 bits)

```
global _start          ; exporta para o ligador (ld) o ponto de entrada

section .text
_start:

    ; sys_write(stdout, mensagem, tamanho)

    mov rax, 1             ; chamada de sistema sys_write
    mov rdi, 1             ; stdout
    mov rsi, mensagem      ; endereço da mensagem
    mov rdx, tamanho       ; tamanho da string de mensagem
    syscall                ; chamada ao núcleo (kernel)

    ; sys_exit(return_code)

    mov rax, 60            ; chamada de sistema sys_exit
    mov rdi, 0             ; retorna 0 (sucesso)
    syscall                ; chamada ao núcleo (kernel)

section .data
mensagem: db 'Hello, world!',0x0A     ; mensagem e quebra de linha
tamanho:  equ $ - mensagem            ; tamanho da mensagem
```

Geração do executável

Passo 1 – Geração do código objeto

- Usando NASM, em um computador de 32 bits:

```
$ nasm -f elf32 hello.asm
```

- Usando NASM, em um computador de 64 bits:

```
$ nasm -f elf64 hello.asm
```

- Usando o GAS:

```
$ as -o hello.o hello.S
```

Os comandos acima gerarão um arquivo `hello.o`.

Geração do executável

Passo 2 – Ligação (geração do código de máquina)

```
$ ld -s -o hello hello.o
```

O comando acima gerará o arquivo executável `hello` .

Desmontadores (*disassemblers*)

- Um desmontador é um programa que recebe como entrada um executável ou código objeto qualquer e gera como saída um programa em linguagem de montagem
- O desmontador que uso em aula é o `objdump`

  Exemplo de uso:

  ```
  $ objdump -D hello.o
  ```

Na verdade, o `objdump` serve para examinar o código objeto como um todo.

Estrutura de um programa

Seções

- `.text` – onde fica o código executável; é uma seção só para leitura
- `.data` – onde fica os dados/variáveis inicializados
- `.bss` – onde fica os dados/variáveis não inicializados
- `_start` – delimita o início do programa principal (comparável à função `main()` em linguagem C)
- `.global _start` – declara o símbolo como visível externamente

Declaração de variáveis

- Variável é um nome simbólico para um dado atualizável pelo programa
- Cada variável possui um tipo e recebe um endereço de memória
- Usa-se pseudo-instruções para definir o tipo da variável
- O montador atribui o endereço de memória
- Nomes de variáveis, constantes e rótulos devem:
  - conter somente letras, números ou os caracteres '_', '$', '.'
  - iniciar por letra, '_', ou '.' (sendo que o uso do ponto denota um rótulo local [1])
- É sensível a letras e maísculas e minúsculas nos nomes dos identificadores

[1] Veja mais sobre rótulos locais na Seção 3.9 do documento em `http://www.nasm.us/xdoc/2.10.07/html/nasmdoc3.html`.

Declaração de variáveis inicializadas

A declaração de variáveis inicializadas é feita na seção `.data` .

"Tipos" de variáveis inicializadas:

Pseudo-Instrução	**Entende-se por**
.byte	define byte (1 byte)
.word	define word (2 bytes consecutivos)
.4byte	define doubleword (4 bytes consecutivos)
.8byte	define quadword (8 bytes consecutivos)

Acessando a memória por meio de variáveis

```
.data

var1:    .word     0x0f17

.text
         mov       $var1,%eax # copia em EAX o endereço de
                              # memória associado a VAR1
         mov       var1, %eax # copia em EAX o valor de VAR1,
                              #ou seja, 0F17H
```

Algumas diferenças entre as sintaxes da AT&T e Intel

	AT&T	**Intel**
Ordem dos operandos	MOV *origem*, *dest*	MOV*dest*, *origem*
Declaração de variáveis	var1: .int *valor*	var2: DB *valor*
Declaração de constantes	const1 = *valor*	const2: EQU *valor*
Uso dos registradores	MOV %eax,%ebx	MOV ebx,eax
Uso das variáveis	MOV $var1,%eax	MOV eax,var2
Uso das constantes	MOV $const1,%eax	MOV eax,const2
Uso de imediatos	MOV $57,%eax	MOV eax,57
Num. hexadecimais	MOV $0xFF,%eax	MOV eax,0FFh
Tam. das operações	MOVB [ebx],%al	MOV al, byte [ebx]
Delimitador de comentários	# comentário AT&T	; comentário Intel

Exercício

Faça um programa em linguagem de montagem que leia um texto da entrada padrão, passe-o para letras maiúsculas e mostre o resultado na saída padrão.
Caracteres que não são letras minúsculas devem permanecer inalterados.

Mais Exercícios (lição de casa)

1. Faça um programa em linguagem de montagem que leia um texto da entrada padrão, inverta-o e mostre o resultado na saída padrão.
2. Faça um programa em linguagem de montagem que leia um arquivo texto e conte o número de caracteres e o número de palavras presentes no arquivo. Considere que o separador de palavras é o caracter de espaço (‘ ’). Obs.: para imprimir os resultados das contagens na saída padrão, você precisará converter um número em *string*.

Bibliografia e materiais recomendados

- Capítulos 3, 4 e 6 do livro *Linux Assembly Language Programming*, de B. Neveln
- Livro *The Art of Assembly Language Programming*, de R. Hyde
  `http://cs.smith.edu/~thiebaut/ArtOfAssembly/artofasm.html`
- *The Netwide Assembler* – NASM
  `http://www.nasm.us/`
- *GNU Assembler* – GAS
  `http://sourceware.org/binutils/docs-2.23/as/index.html`
- *Linux assemblers: A comparison of GAS and NASM*
  `http://www.ibm.com/developerworks/linux/library/l-gas-nasm/index.html`
- Tabela de chamadas ao sistema no Linux
  `http://www.ime.usp.br/~kon/MAC211/syscalls.html`

Cenas dos próximos capítulos...

- Ainda sobre linguagem de montagem
  - Uso da pilha de dados
  - Subrotinas